

SPECIFICATION

Electronic Version 1.2.8

Stylesheet Version 1.0

METHOD FOR THE EFFICIENT COMPRESSION OF GRAPHIC CONTENT IN COMPOSITE PDF FILES

Related Applications

Referenced-applications

This application claims the benefit of U.S. Provisional Application Serial Number 60/253,812, filed on November 29, 2000.

Field of the Invention

This invention relates to the use of Adobe ® Acrobat ® PDF files, and, in particular, discloses a method for creating a compressed form of Adobe ® Acrobat ® PDF containing composite PDF pages.

Background of Invention

[0001] The Adobe ® company's Acrobat ® line of products is designed to allow the creation and viewing of documents in a platform-independent manner. Files can be created or converted to Adobe ® Acrobat ® Portable Document Format (PDF), which is an open de facto standard for electronic document distribution. PDF is a universal file format that preserves all the fonts, formatting, graphics, and color of the source document, regardless of the application and platform used to create it. Adobe PDF files can be shared, viewed, navigated, and printed exactly as intended. The PDF format and the line of Acrobat ® products provides platform transportability for documents.

[0002] Each PDF file can describe multiple pages of content. Within each PDF file, the

pages are represented in a database having the form of a tree-structure. Each PDF page is accessible from the well-defined root of the tree-structure using a known traversal algorithm. The graphic content of each page, in the form of Cos objects, is represented by one or more terminal nodes in the structure.

[0003] In the publishing industry, it is useful to be able to produce composite pages which consist of multiple component graphic objects. It is not uncommon for some of the graphic objects to be variable in nature, such as, for example, the addressee on a form letter. It is also useful and convenient for the component graphic objects to be stored as static PDF files.

[0004] Figure 1 shows a process by which component static PDF files composed of graphical components are composited onto a single page or multiple pages to create a final document. Figures 1A & 1B shows an individual composite page. The page contains a letter in which the logo, letter body and signature block can be varied among several choices, most likely depending upon the recipient of the letter. The variable portions of the letter are stored in static PDF files. Figure 1(C) shows the selection of variable graphic objects for placement of the page based on a variable data stream containing the addressee information. Lastly, Figure 2 shows the retrieval of the variable graphic objects from static PDF files.

[0005] Traditionally this process has been accomplished by RIPing (sending through a Raster Image Processor) each PDF file and using assembling hardware and/or software to overlay the raster images of each component to create a composite page. This construction is driven by a static, descriptive language, called a compositing language, that defines the appearance of each page in terms of component pages.

[0006] Because the RIP process can be computationally and time intensive, a technique is needed to organize a PDF file so that 1) multiple RIPs can be efficiently and simultaneously used to rasterize the file and 2) component PDF files that are used multiple times, i.e., composited more than once onto a single page or different pages, can share a single pass through the RIP and/or rasterization process.

Summary of Invention

[0007] A technique for creating a compressed form of Adobe® Acrobat® PDF from a

stream of variable page descriptions and static PDF files is disclosed. The resulting compressed files are compatible with the Adobe[®] Acrobat[®] line of products. The technique includes a symmetrical compression/decompression process yielding a file in which composite pages and their components can be efficiently and directly accessed.

Brief Description of Drawings

- [0008] [01]Figure 1 shows how complex documents are assembled from multiple, components PDF files.
- [0009] Figure 1(A) shows a complex document having variable fields.
- [0010] Figure 1(B) shows variable data being placed into the document of Figure 1(A).
- [0011] Figure 1(C) shows the selection of components for the variable data fields based on a data stream.
- [0012] Figure 2 shows the selection of components for the variable data fields from a PDF file.
- [0013] Figures 3(A–C) show the placement of Cos objects and allocated storage blocks in a compressed composite PDF file.
- [0014] Figure 4 shows the links between allocated storage blocks, contents arrays and Cos objects.

Detailed Description

- [0015] The following process describes the construction of multiple component static PDF files into a composite, compressed file which is compatible with the Adobe[®] Acrobat[®] line of products. Several assumptions are made regarding the static PDF source files. First, each component PDF page in a static source PDF file is modified at most by one clipping function and a single 2-dimensional matrix transformation (CTM or Current Transformation Matrix). The clipping function describes an area of the page which is to be eliminated from the composite (i.e., a hole in the page) such that other pages underlying the clipped page can show through. The matrix transformation is a mathematical function that rotates, scales or offsets an object.

Second, each static PDF file has a contents array containing references to PDF commands and a resources dictionary that identifies all resources used on the page.

[0016] The process that describes the construction of each composite page is defined by a compositing language, several of which are well known in the art. Some examples include ASCII text, pdfExpress script, PPML and the Barco Book Ticket Language. Using one of these languages, each resulting composite page is described separately. Within the description of each composite page, every placement of a PDF page from a static PDF file is defined by its own CTM and clipping function. Further, on each composite page, the placement of a static page, using the clipping function and the CTM, overlays any prior marks caused by previous placements on the composite page. Using the compositing language, the composite page description can be parsed efficiently into a single, random access data structure representing the entire assembly process. The compositing language may also support other features, such as the substitution of text or images.

[0017] The compressed output composite PDF file is first defined by a sequence of operations, based on the compositing language, which are recorded for later analysis. The recording of the operations allows an analysis and optimization step to occur prior to the creation of the output file. The compression is a two-phased operation. In the first phase, components of the input PDF files needed in the composited output are analyzed. These components may need to be modified in some manner. In the second phase the components are copied, and any necessary modifications are applied.

[0018] Prior to the analysis, a sequence of operations that select and place PDF pages from existing static PDF files into new pages in the output PDF file is recorded in a data structure. Each new composite PDF page is defined by the following:

[0019] 1. A base page size defined by a rectangle;

[0020] 2. A base PDF rotation for the base page defining the orientation of that page (i.e., 0°, 90°, etc.); and

[0021] 3. Zero or more additional PDF pages layered or overlaid on top of the base page. For each such page, a CTM and a clipping function is specified.

[0022] In phase I of the analysis, each input static PDF file is opened and the PDF pages within the files are located, using the root and tree-structure. For each page, the PDF resources (Cos objects) that are used on that page are identified and assigned a unique name. A resource lookup table is created to associate the unique name with the original name of the resource. Resources shared across two or more PDF pages are tracked separately for each page that references them. Next, the PDF content stream for each page is examined to determine all points where a PDF operator references a resource. The information required to modify the reference to the resource is stored. Lastly, certain information about the PDF component parts (i.e., PDF Cos objects and PDF content streams) of each page is stored in a second table associated with each PDF file and each page. This table defines a set of modifications to the input PDF files that allow the contents of the files to be used in the next phase.

[0023] In phase II, the output file is constructed. First, an output file usage table is created which has an entry for each PDF component part (i.e., PDF content stream or PDF Cos object). Initially, all entries are set to FALSE, indicating that the component has not been copied to the output file. Next, the recorded operations are sequentially processed. When a reference to a particular PDF page is encountered within the recorded operations, the output file usage table is tested for each component part of the page. If a FALSE entry is encountered, the component part is copied to the output file and the table is set to TRUE. Conversely, if a TRUE entry is encountered, no copying is performed. The effect is to copy each PDF component part to the output file only one time. The copied PDF components are shown in Figure 3B. As each object is copied to the output file, it is assigned a new ID number, such that a new numbering system is created only for objects copied to the output file. As PDF content streams are encountered, each operator is inspected. If a reference is made to an original PDF resource, a lookup is performed in the resource lookup table, and the operation is modified to refer to the resource using the new, unique name of the resource instead of the original name.

[0024] For each new output page, a new PDF content stream is created to represent the page. Preferably, the new content stream is constructed as a Cos array type object (Contents Array), but other representations are possible. In the output file, a static block of storage is allocated for each new output page. This is shown in Figure 3(C).

The storage block links resources and the Contents Array for each page. The Contents Array is populated with references to the Cos objects using the IDs that were assigned when the objects were copied to the output file.

[0025] For each static PDF page to be composited onto a new output page, the following occurs: A PDF stream consisting of commands which terminate and initiate graphic isolation, transform and crop the static PDF page, and identify the page is constructed. The page identification consists of the following information: a platform independent path and file name, the original page number, the total number of pages in the original file, a unique checksum dependent on the content streams representing the page, the clipping function, the CTM, and the number of times this file/page combination is used (defined as use count). This stream is converted to a Cos object and inserted in the next available Contents Array slot. The Contents Array elements for the page are then copied into the array and the next available slot is advanced to the first available slot after the copied contents. Finally, as shown in Figure 4, the Contents Array is written to the output file, along with the information necessary to reference the proper resources required for that page.

[0026] In alternative embodiments, un-referenced static PDF pages and un-used Cos objects, as well as the text of the original compositing language, may be stored in the PDF files as storage blocks for purposes of reconstruction if the file is to be decompressed.

[0027] The compressed file can be decompressed in a reverse process. The compressed composite PDF is opened and a traversal of all of the Contents Array for each of the composite PDF pages in the file, as well as the list of unused static PDF pages is completed. All references to the original, static PDF pages are collected and sorted by file name and page number, and duplicate entries are removed. During the traversal a compositing language stream is constructed based on the file names, page numbers, CTMs, clipping functions and composite page structure. This language duplicates the original used to construct the file. Alternatively, the original language could be extracted if stored as described above.

[0028] The list is processed by file name. The PDF files are constructed using the page components identified in the traversal described above.

[0029] Random access to page information is driven by the PDF page structure. For a given page, the PDF page structure is traversed to locate the page. This yields approximately $\text{Log}(n)$ performance for accessing pages (where n is the number of pages) because the PDF Page structure acts as a database-style index to the given pages.

[0030] Because the PDF file represents the entire set of knowledge required for a complete, set-theoretic job decomposition of itself, the RIPing process can make a complete, static analysis of the file. This static analysis can be used to efficiently process the file, and the process can be optimized based on the particular production environment (i.e., output device).

[0031] It can be seen that various modifications could be made to the above-described process without deviating from the spirit or scope of the invention, which is embodied in the claims below. For example, the content copying pass and page construction pass could be combined into a single pass.

TOPT-EE9960